SvPablo and the PERC Performance Tool Suite

Evan Welbourne, NERSC/LBNL

NUG Meeting, June 4th, 2002

Credit / References

Daniel A. Reed, Ruth A. Aydt, Luiz DeRose, Celso L. Mendes, Randy L. Ribler, Eric Shaffer, Huseyin Simitci, Jeffrey S. Vetter, Daniel R. Wells, Shannon Whitmore, and Ying Zhang, "*Performance Analysis of Parallel Systems: Approaches and Open Problems*," Joint Symposium on Parallel Processing (JSPP), Nagoya, Japan, June 1998 (invited paper and keynote presentation), pp, 239-256.

PERC consortium, *The Original Proposal for the PERC Project.*
Proposal in response to the DOE SciDAC Solicitation 01-07, Mar. 9, 2001

PERC consortium, *The Performance Evaluation Research Center (PERC).*
slides from the poster session at the SciDAC PI meeting, Jan. 15-16, 2002

PERC consortium, personal contact

Slides 13-38 adapted from:

Celso Mendes, Ying Zhang, Dan Reed,
SvPablo: A Toolkit for Performance Analysis of Parallel Systems.
slides from tutorial given at the PERC SciDAC meeting during SC2001,
Nov. 2002

The Changing face of High Performance Computing

As the science and technology of high performance computing advances, our applications and the environment in which they execute is constantly evolving.

- Larger systems: HPC systems are scaling to a larger and larger numbers of processors.
- Complex codes: Codes are often multi-language, and written using object-oriented, and data-parallel programming languages.
- Diverse environment: With the advent of the computational grid, our codes will no longer execute on homogeneous systems, but on a large heterogeneous system.

✍ Application performance is a complex function of many variables; it's often counter-intuitive, and probably not predictable using first-principles. Our performance tools are not suited for the future of HPC.

Requirements for the Success of a Modern Performance Toolkit

To succeed in the rapidly evolving world of HPC, a performance toolkit should be:

- ✍ Scalable: Tools should scale for use on large systems.
- ✍ Portable: Tools should work on every system in the grid.
- ✍ Compatible: Tools should be work with most programming languages.
- ✍ Versatile: Tools should leverage a variety of HW and SW techniques.
- ✍ Experimental: Since analysis from first-principles is unlikely.

As always, we would like our performance tools to be:

- ✍ Easy to use: Simple and intuitive for users.
- ✍ Non-intrusive: Disruption of the normal execution and usage pattern for an application should be minimal.

- ✍ The PERC project aims to develop a toolkit with these qualities.

The PERC SciDAC Project

The Performance Evaluation Research Center (PERC) is an "Integrated Software Infrastructure Center" (ISIC) sponsored under DoE's SciDAC program.

- Funding: approx. $2.4 million per year.

- Mission:

  Develop a science of performance.

  Engineer tools for performance analysis and optimization

- Focus:

  Large, grand-challenge calculations, especially SciDAC application projects.

PERC website: http://perc.nersc.gov

PERC Participants

PERC is a collaboration among eight institutions: four DoE laboratories, and four Universities.

Lawrence Berkeley National Lab David Bailey Erich Strohmaier	**University of Tennessee** Jack Dongarra
Argonne National Lab Paul Hovland Boyana Norris	**University of Illinois at Urbana-Champaign** Dan Reed
Lawrence Livermore National Lab Dan Quinlan Bronis de Supinski Jeffery Vetter	**University of Maryland at College Park** Jeff Hollingsworth
Oak Ridge National Lab Pat Worley Tom Dunigan	**San Diego Supercomputing Center** Allan Snavely

The PERC project is led by David H. Bailey at NERSC/LBNL.

PERC's tool development effort is led by Dan Reed at UIUC.

The PERC Performance Tool Suite

The goal of PERC's tool effort is to produce an interoperable suite of measurement, analysis, and tuning tools that are suited for use on current and future HPC systems.

This goal requires three tightly coupled research efforts:

- End-user tools that integrate various analysis and measurement approaches, providing a common interface for comparing performance measurements across platforms and executions and correlating this data with benchmark and application source code.
- Flexible instrumentation systems for capturing hardware and software interactions, instruction execution frequencies, memory reference behavior, and execution overheads.
- Data management infrastructure for tracking performance experiments and data across time and space.

PAPI – Performance API

An API for portable hardware measurement

Provides the tool designer with a consistent interface and methodology for use of the performance counter hardware found in most major microprocessors.

PAPI is available for Linux/x86, Windows 2000, Linux/IA-64, Sun Solaris/Ultra 2.8, IBM AIX/Power, SGI IRIX/MIPS, Compaq Tru64/Alpha Ev6 & Ev67, and Cray T3E/Unicos.

To use PAPI on Seaborg, issue the command:

```
% module load papi
```

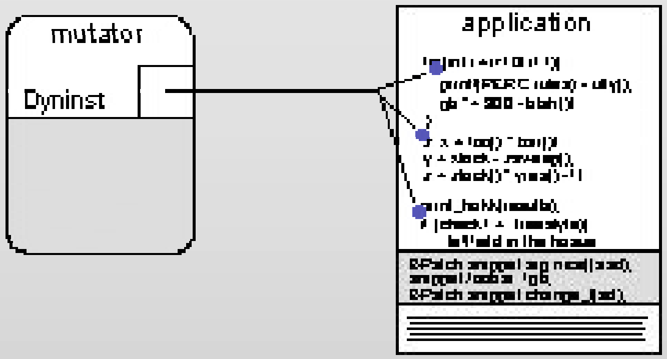
For more information on PAPI at NERSC, see the NERSC help page:
http://hpcf.nersc.gov/software/tools/papi.html

PAPI project homepage: http://icl.cs.utk.edu/projects/papi/

Dyninst API

An API for dynamic instrumentation at runtime.

Provides a machine independent interface to permit the creation of tools that use runtime code patching.



The interface is analogous to a machine independent intermediate representation of instrumentation as an abstract syntax tree.

The Dyninst API is available for MIPS (IRIX), Power/PowerPC (AIX), SPARC (Solaris), and x86 (Linux, Solaris and NT).

Dyninst project homepage: http://www.dyninst.org

Performance Bounds

A set of tools for establishing bounds on the performance of an application or program construct.

Will use source code analysis to determine what sections of code are memory bandwidth limited, instruction scheduling limited, etc. on a given architecture.

The tools will also utilize optional user annotations in order to provide more accurate bounds for performance-critical sections of code

Sigma

A tool for memory hierarchy measurement.

Uses runtime instrumentation to extract a detailed representation of the memory reference pattern of an application.

The memory reference pattern information will be the input to a collection of post-execution tools that provide insight into memory performance issues such as cache conflicts and memory bandwidth contention.

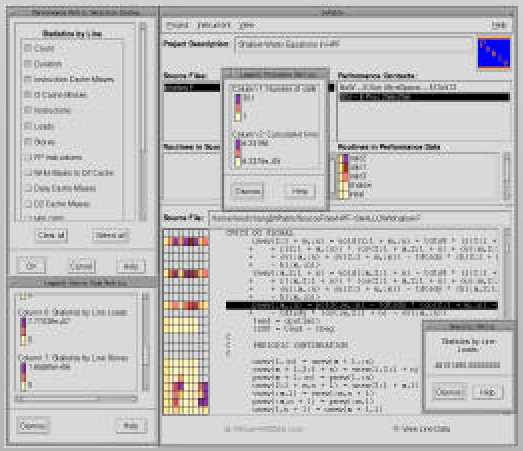
The Sigma effort is a joint collaboration between IBM and the University of Maryland.

SvPablo

An end-user tool that supports source instrumentation and browsing of runtime performance data with a graphical user interface.

Incorporates the APIs and other performance tools to provide a front-end to the PERC tool suite.

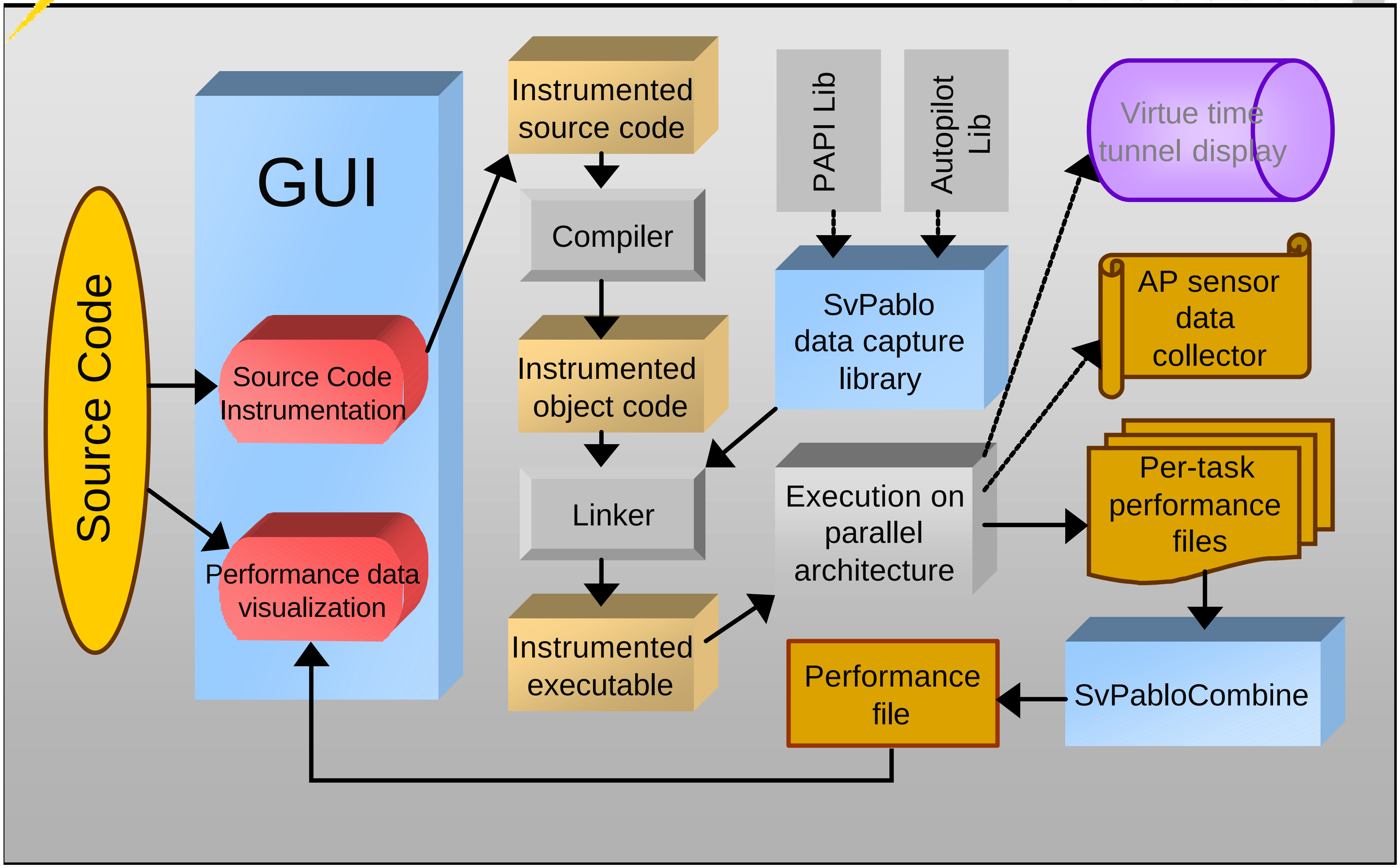
SvPablo is available for: SunOS 5.7, SGI Irix 6.5, IBM SP2 AIX 4.3, RedHat Linux 6.1, 6.2, 7.1, Intel Itanium IA-64/RedHat 7.1, and Compaq Alpha OSF1 5.1



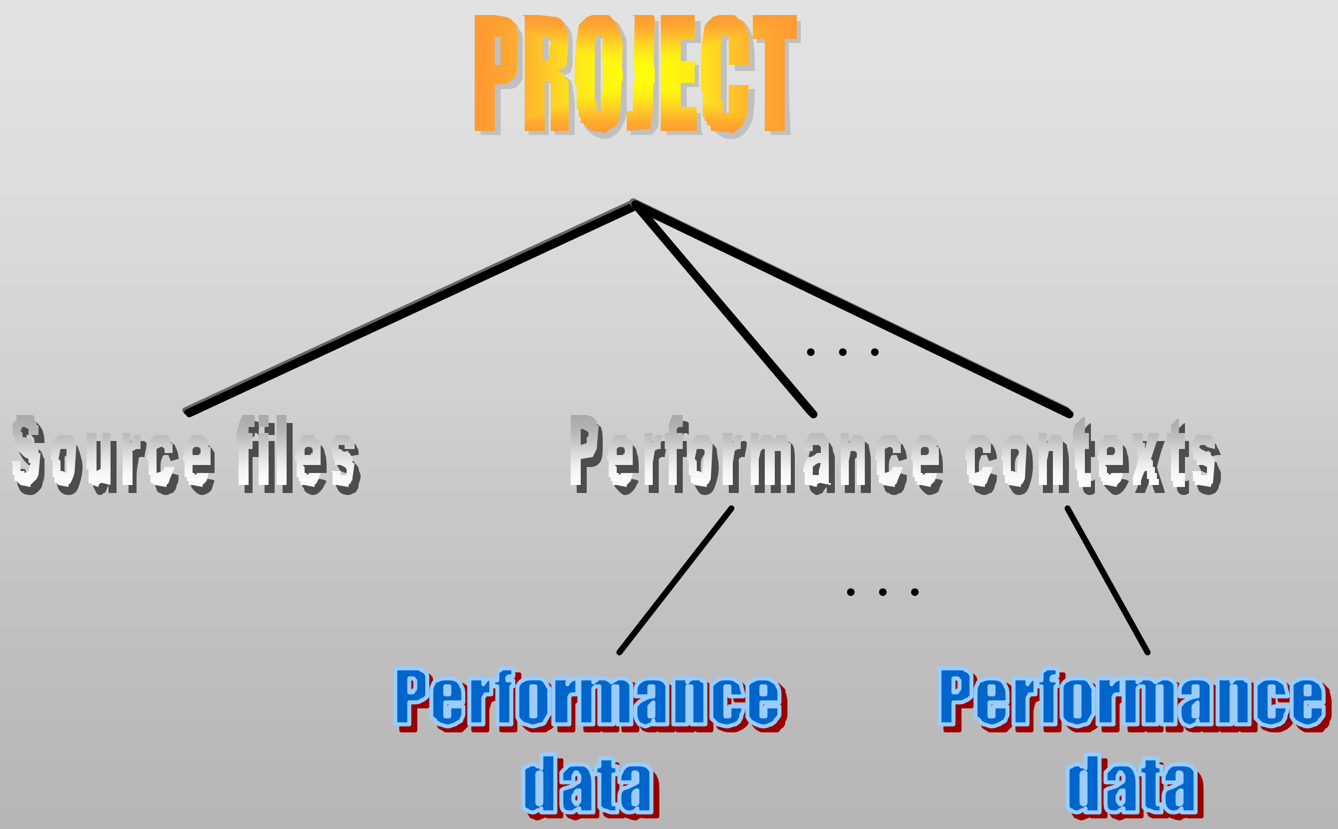
Supports C, Fortran-77, Fortran-90 (free and fixed form), HPF, and MPI

SvPablo project homepage:
http://www-pablo.cs.uiuc.edu/Software/SvPablo/svPablo.htm

SvPablo Components



SvPablo Model



Using SvPablo at NERSC

SvPablo v5.1 is installed on NERSC's Seaborg machine.

To use SvPablo on Seaborg, issue the command:

```
% module load svpablo
```

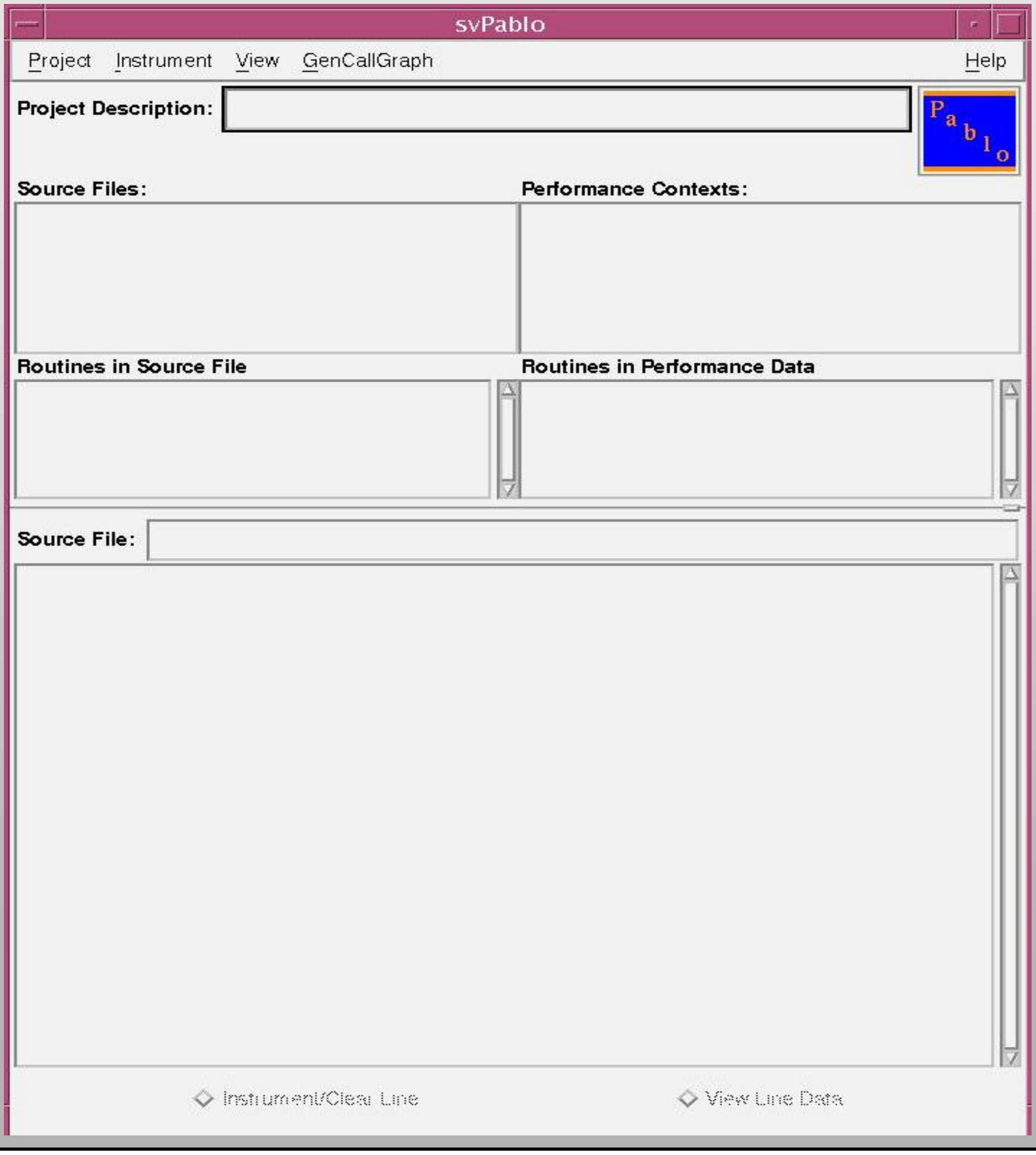
SvPablo's main window can then be launched with the command:

```
% runSvPablo &
```

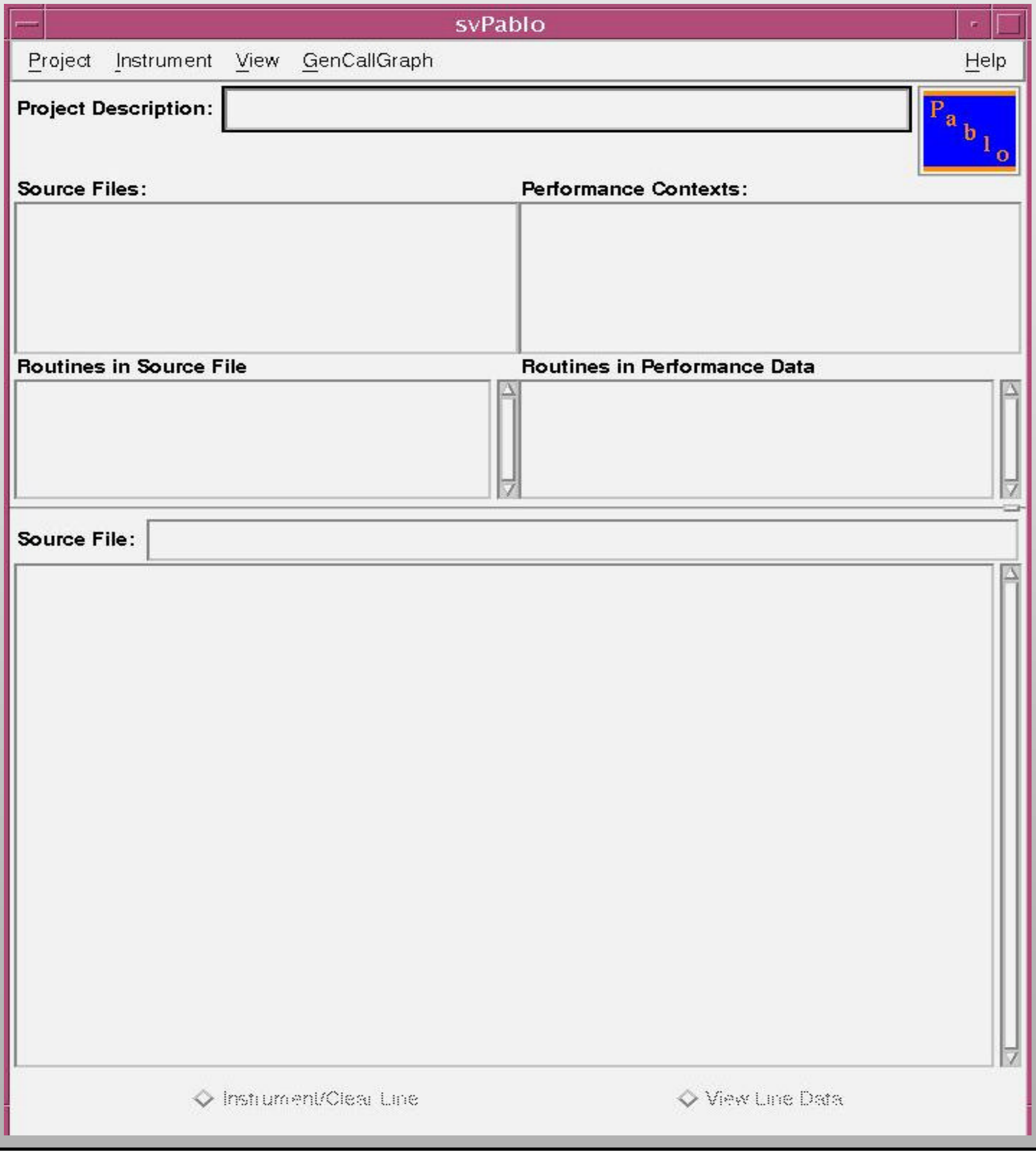
The **SvPabloCombine** command may be issued on the command line as well:

```
% SvPabloCombine  <parameters>
```

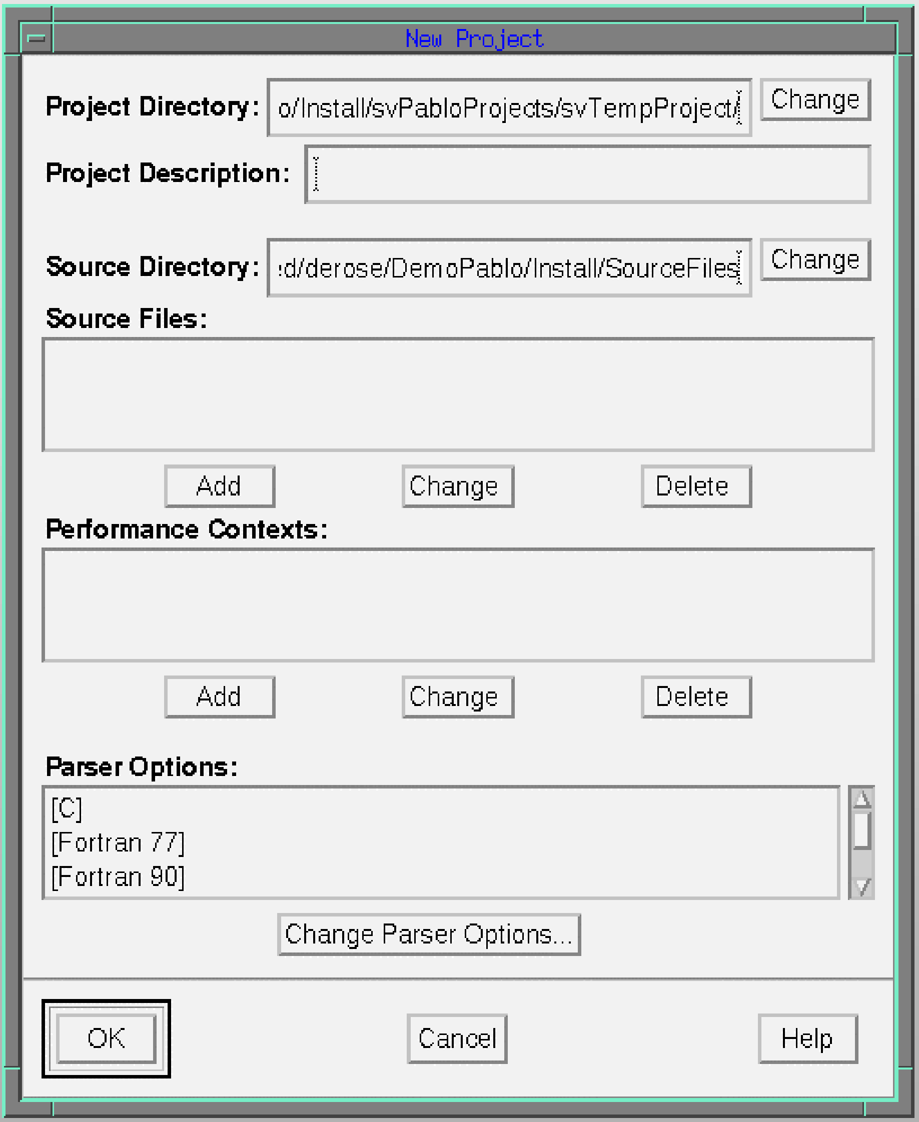
The SvPablo Main Window



Project Menu



New Project Dialog Box



New Project

Project Directory: o/Install/svPabloProjects/svTempProject/

Project Description:

Source Directory: :d/derose/DemoPablo/Install/SourceFiles

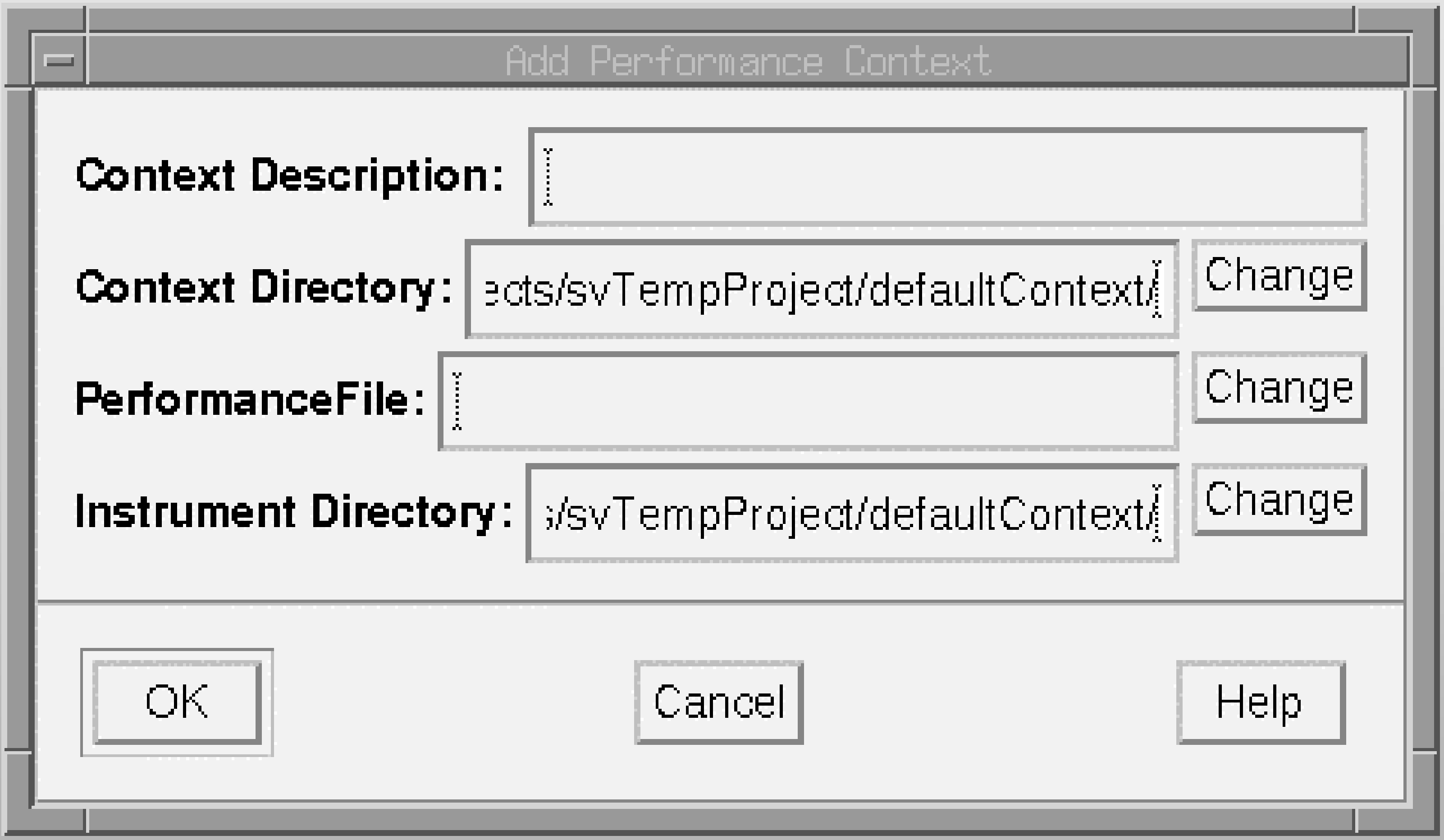
Source Files:

Performance Contexts:

Parser Options:

[C]
[Fortran 77]
[Fortran 90]

Performance Context Dialog Box


Add Performance Context

Context Description:

Context Directory: ects/svTempProject/defaultContext/ [Change]

PerformanceFile: [Change]

Instrument Directory: s/svTempProject/defaultContext/ [Change]

OK Cancel Help

Interactive Instrumentation and Analysis

Typically done in three steps:

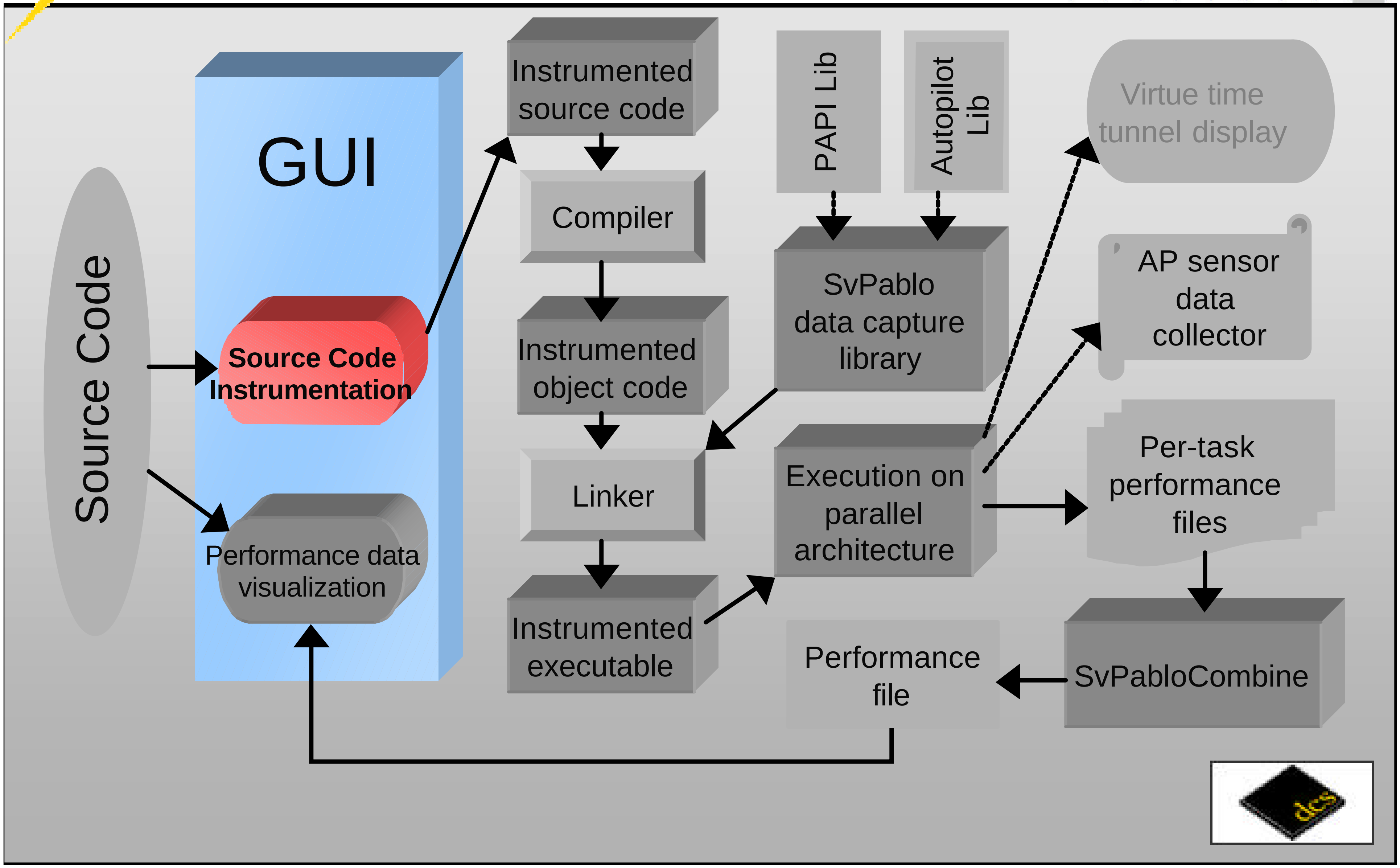
1. source code instrumentation
2. program compilation and execution
3. performance data visualization

Each cycle (1-2-3) corresponds to a Performance-Context

If desired, the cycle can be repeated (multiple performance-contexts)

Steps 1 and 3 are done in the GUI

Step 1: Source Code Instrumentation

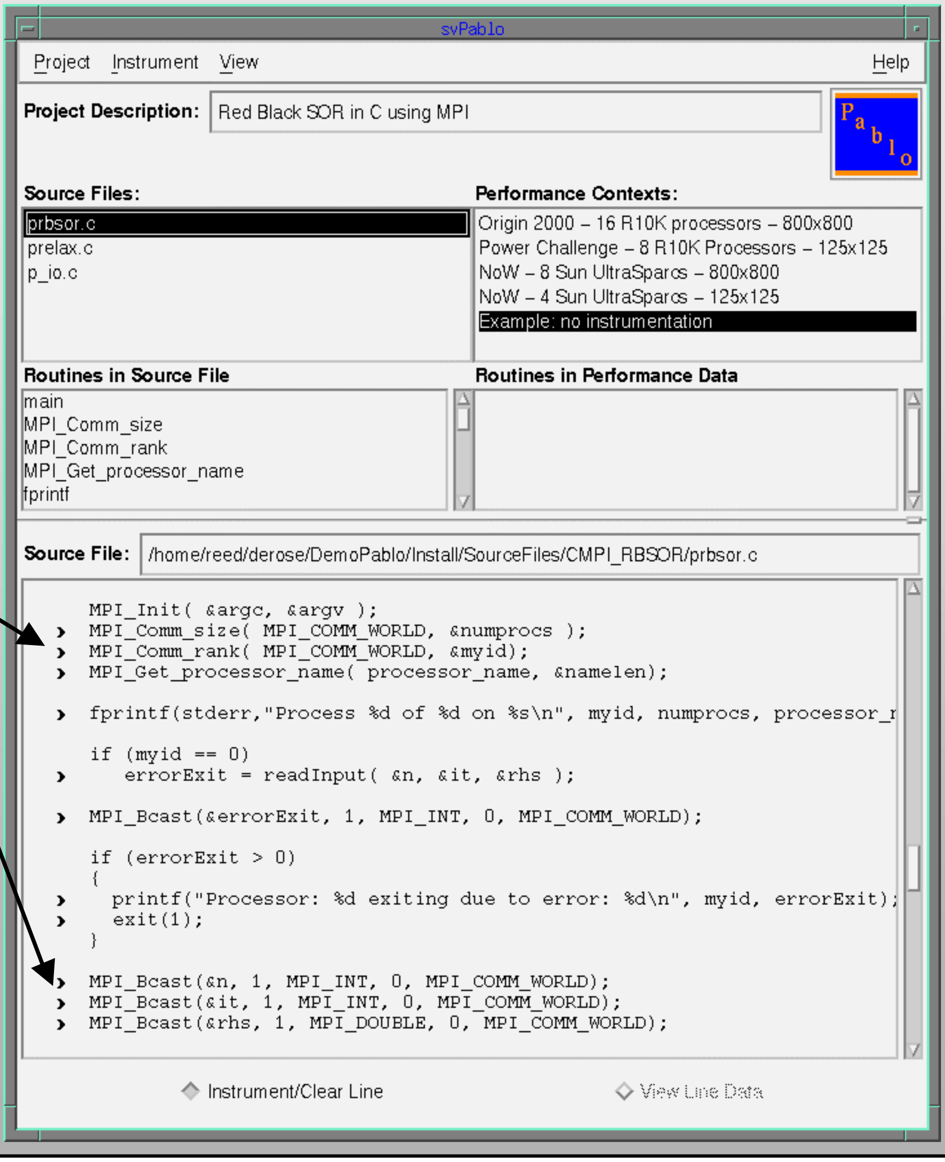


Source Code Instrumentation

- Instrumentable constructs:
  - ✍ procedure calls
  - ✍ outer loops
- Basic metrics:
  - ✍ counts
  - ✍ inclusive durations
  - ✍ exclusive durations
- Optional metrics:
  - ✍ any metric provided by PAPI

Interactive Instrumentation

instrumentable constructs (function calls and outer loops)



Project Description: Red Black SOR in C using MPI

Source Files:
prbsor.c
prelax.c
p_io.c

Performance Contexts:
Origin 2000 – 16 R10K processors – 800x800
Power Challenge – 8 R10K Processors – 125x125
NoW – 8 Sun UltraSparcs – 800x800
NoW – 4 Sun UltraSparcs – 125x125
Example: no instrumentation

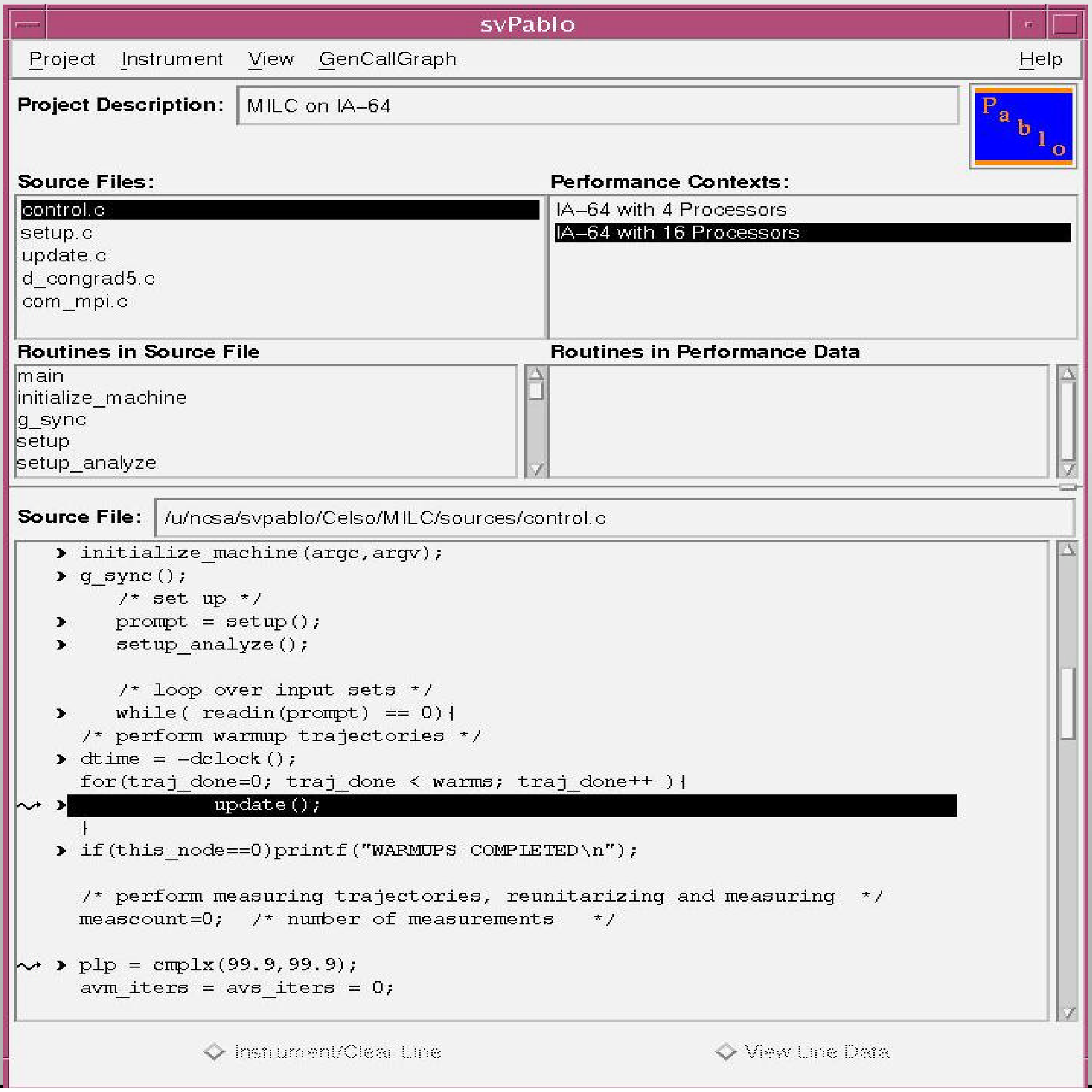
Routines in Source File
main
MPI_Comm_size
MPI_Comm_rank
MPI_Get_processor_name
fprintf

Routines in Performance Data

Source File: /home/reed/derose/DemoPablo/Install/SourceFiles/CMPI_RBSOR/prbsor.c

```
    MPI_Init( &argc, &argv );
 >  MPI_Comm_size( MPI_COMM_WORLD, &numprocs );
 >  MPI_Comm_rank( MPI_COMM_WORLD, &myid);
 >  MPI_Get_processor_name( processor_name, &namelen);

 >  fprintf(stderr,"Process %d of %d on %s\n", myid, numprocs, processor_r

    if (myid == 0)
 >     errorExit = readInput( &n, &it, &rhs );

 >  MPI_Bcast(&errorExit, 1, MPI_INT, 0, MPI_COMM_WORLD);

    if (errorExit > 0)
    {
 >    printf("Processor: %d exiting due to error: %d\n", myid, errorExit);
 >    exit(1);
    }

 >  MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
 >  MPI_Bcast(&it, 1, MPI_INT, 0, MPI_COMM_WORLD);
 >  MPI_Bcast(&rhs, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

Line by Line Instrumentation



Project Description: MILC on IA–64

Source Files:
control.c
setup.c
update.c
d_congrad5.c
com_mpi.c

Performance Contexts:
IA–64 with 4 Processors
IA–64 with 16 Processors

Routines in Source File
main
initialize_machine
g_sync
setup
setup_analyze

Routines in Performance Data

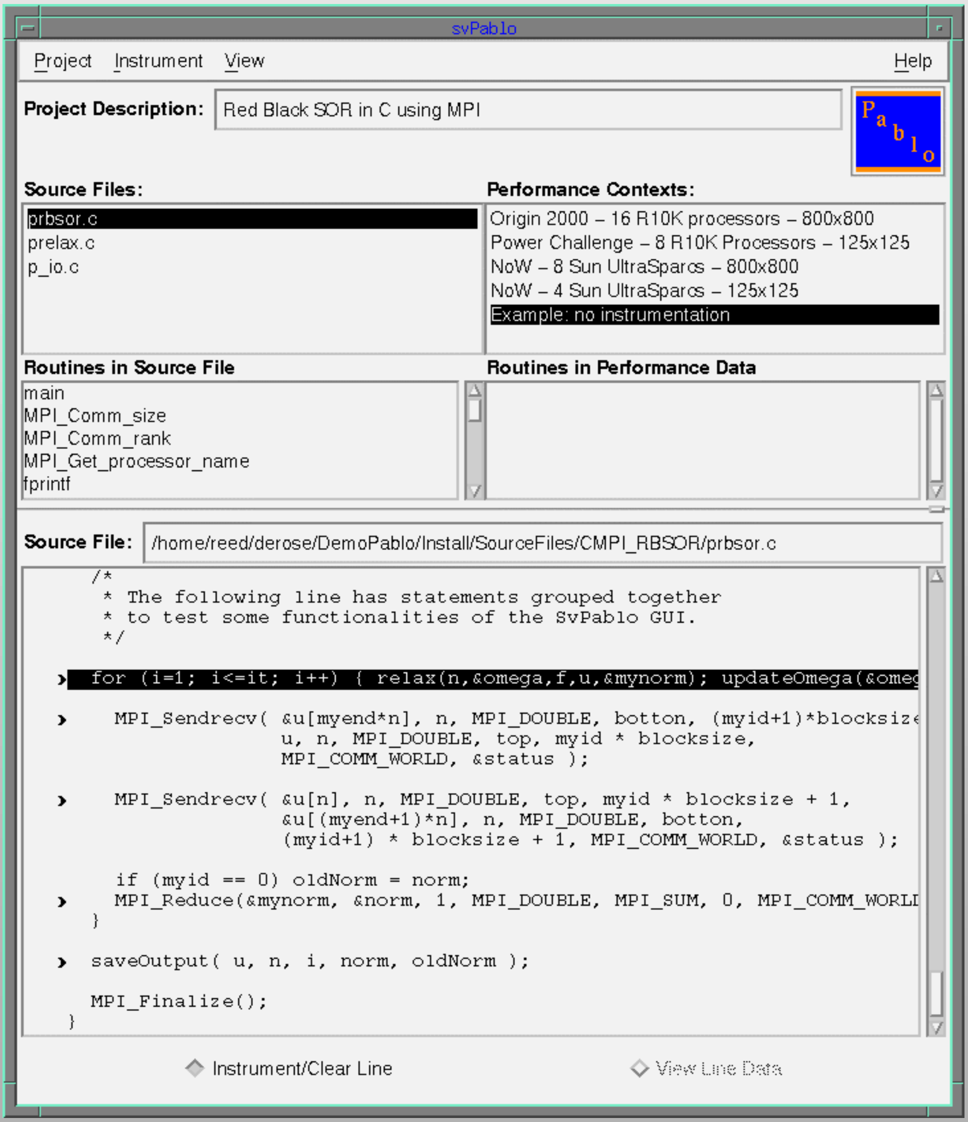
Source File: /u/ncsa/svpablo/Celso/MILC/sources/control.c

```
  > initialize_machine(argc,argv);
  > g_sync();
         /* set up */
  >    prompt = setup();
  >    setup_analyze();

         /* loop over input sets */
  >    while( readin(prompt) == 0){
    /* perform warmup trajectories */
  > dtime = -dclock();
    for(traj_done=0; traj_done < warms; traj_done++ ){
~ >             update();
    }
  > if(this_node==0)printf("WARMUPS COMPLETED\n");

    /* perform measuring trajectories, reunitarizing and measuring  */
    meascount=0;  /* number of measurements   */

~ > plp = cmplx(99.9,99.9);
    avm_iters = avs_iters = 0;
```

Instrumentable lines

Instrumented lines

Line Instrumentation – Multiple Events



Project Description: Red Black SOR in C using MPI

Source Files:
prbsor.c
prelax.c
p_io.c

Performance Contexts:
Origin 2000 – 16 R10K processors – 800x800
Power Challenge – 8 R10K Processors – 125x125
NoW – 8 Sun UltraSparcs – 800x800
NoW – 4 Sun UltraSparcs – 125x125
Example: no instrumentation

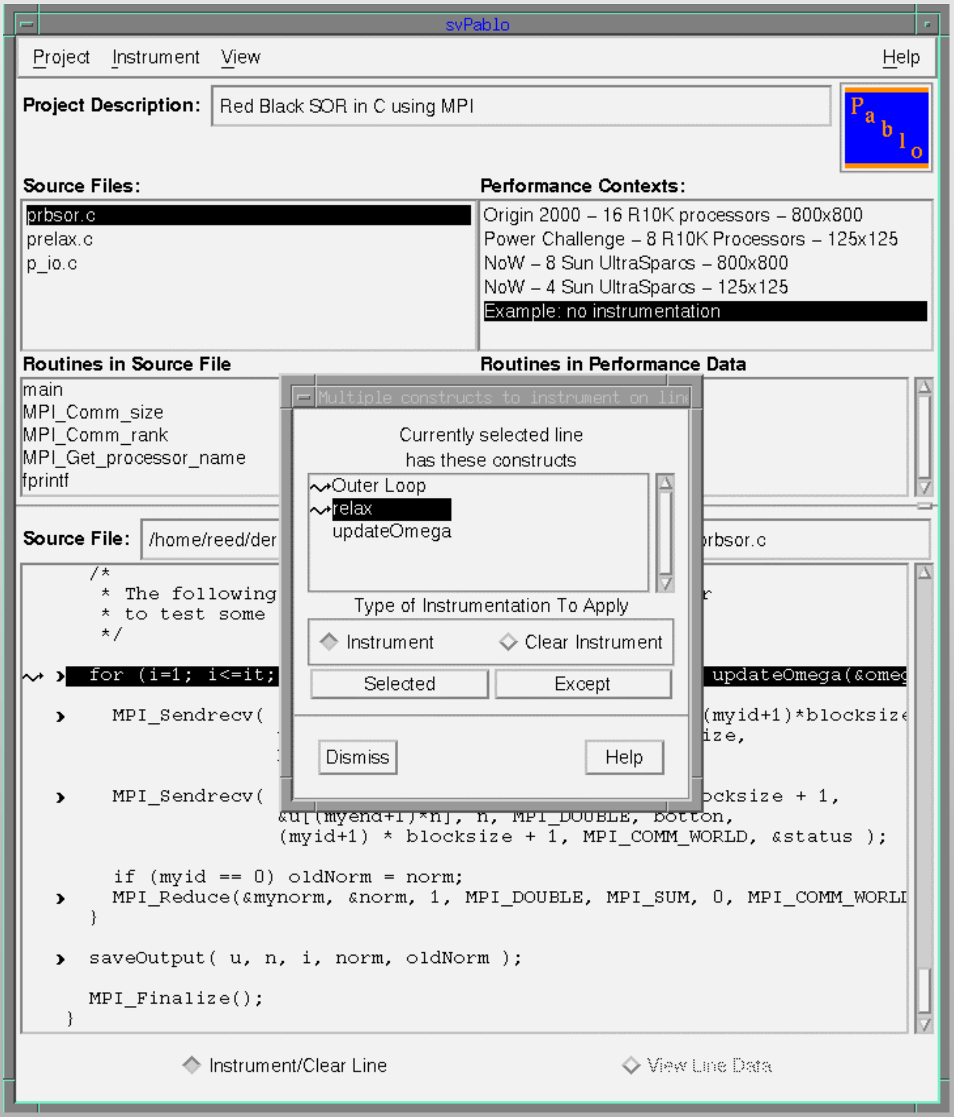
Routines in Source File
main
MPI_Comm_size
MPI_Comm_rank
MPI_Get_processor_name
fprintf

Routines in Performance Data

Source File: /home/reed/derose/DemoPablo/Install/SourceFiles/CMPI_RBSOR/prbsor.c

```
    /*
     * The following line has statements grouped together
     * to test some functionalities of the SvPablo GUI.
     */

›   for (i=1; i<=it; i++) { relax(n,&omega,f,u,&mynorm); updateOmega(&omeg

›     MPI_Sendrecv( &u[myend*n], n, MPI_DOUBLE, botton, (myid+1)*blocksiz
                    u, n, MPI_DOUBLE, top, myid * blocksize,
                    MPI_COMM_WORLD, &status );

›     MPI_Sendrecv( &u[n], n, MPI_DOUBLE, top, myid * blocksize + 1,
                    &u[(myend+1)*n], n, MPI_DOUBLE, botton,
                    (myid+1) * blocksize + 1, MPI_COMM_WORLD, &status );

      if (myid == 0) oldNorm = norm;
›     MPI_Reduce(&mynorm, &norm, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLI
    }

›   saveOutput( u, n, i, norm, oldNorm );

    MPI_Finalize();
  }
```

Line Instrumentation – Multiple Events



Project Description: Red Black SOR in C using MPI

Source Files:
- prbsor.c
- prelax.c
- p_io.c

Performance Contexts:
- Origin 2000 – 16 R10K processors – 800x800
- Power Challenge – 8 R10K Processors – 125x125
- NoW – 8 Sun UltraSparcs – 800x800
- NoW – 4 Sun UltraSparcs – 125x125
- Example: no instrumentation

Routines in Source File
- main
- MPI_Comm_size
- MPI_Comm_rank
- MPI_Get_processor_name
- fprintf

Routines in Performance Data

Source File: /home/reed/der … rbsor.c

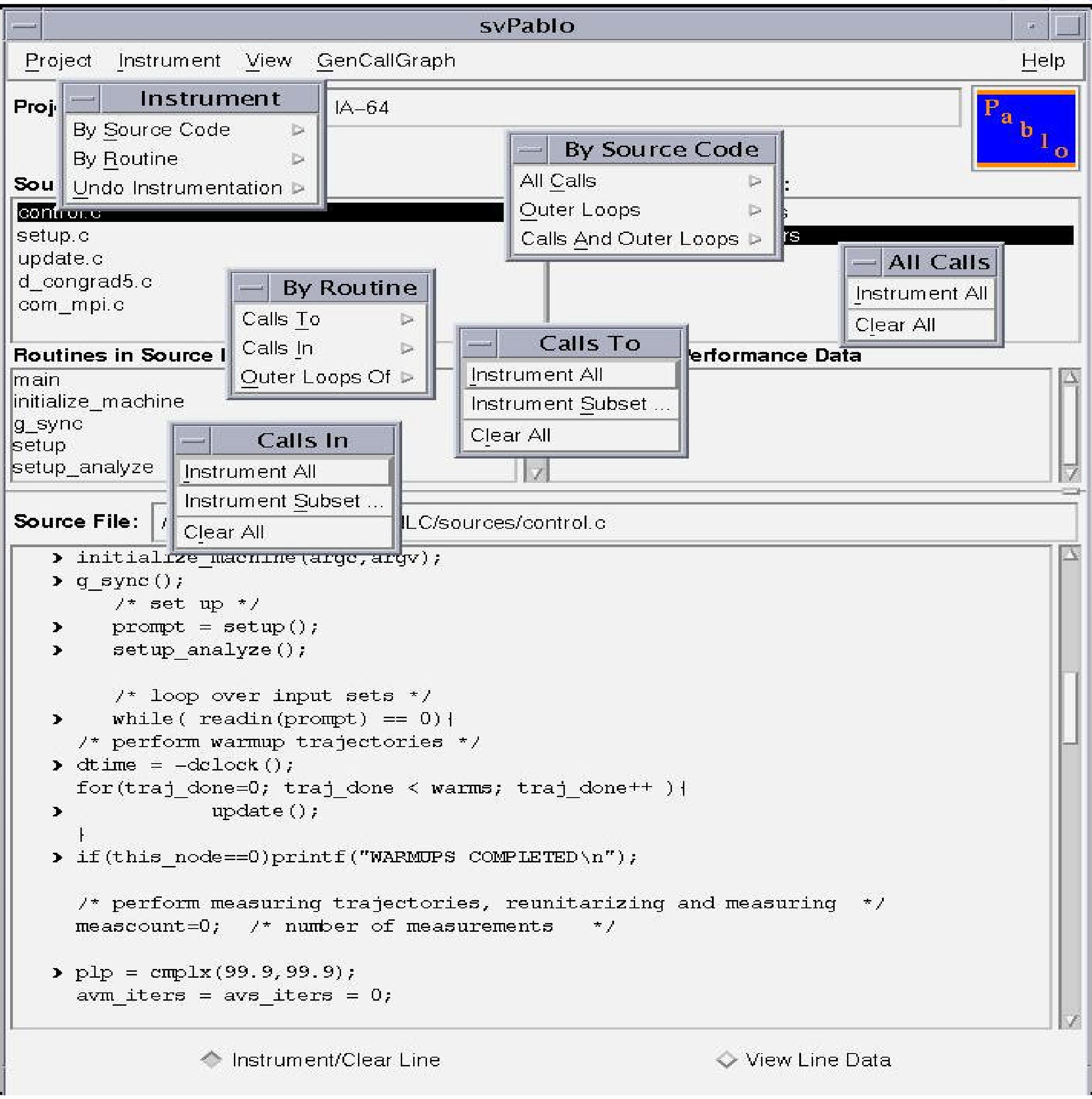
```
     /*
      * The following                                   r
      * to test some
      */

   for (i=1; i<=it;                                  updateOmega(&omeg

     MPI_Sendrecv(                                   (myid+1)*blocksize
                                                     ize,

     MPI_Sendrecv(                                   ocksize + 1,
                    &u[(myend+1)*n], n, MPI_DOUBLE, botton,
                    (myid+1) * blocksize + 1, MPI_COMM_WORLD, &status );

     if (myid == 0) oldNorm = norm;
     MPI_Reduce(&mynorm, &norm, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLI
   }

   saveOutput( u, n, i, norm, oldNorm );

   MPI_Finalize();
 }
```

Multiple constructs to instrument on line

Currently selected line has these constructs
- Outer Loop
- relax
- updateOmega

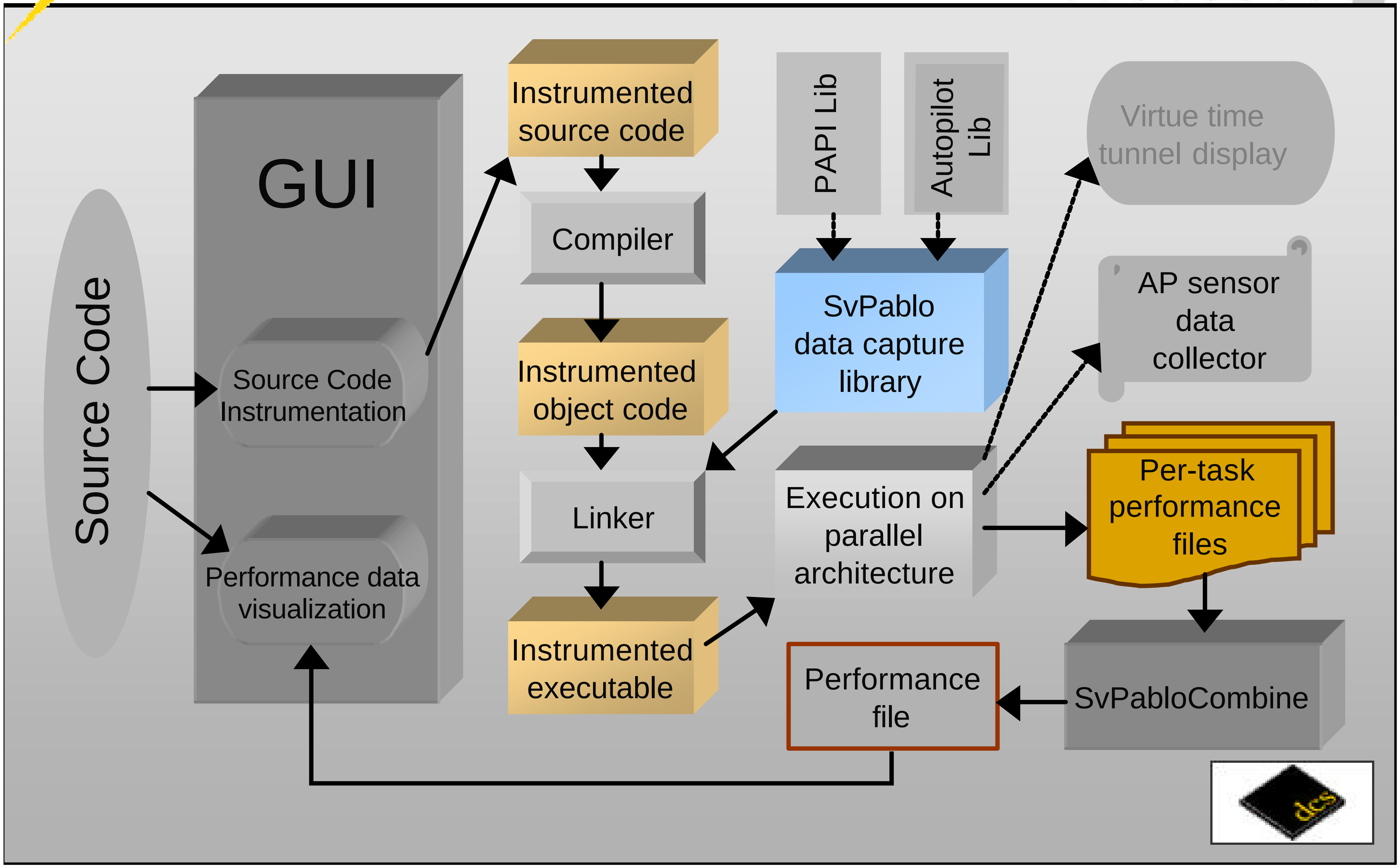
Type of Instrumentation To Apply

Automatic Instrumentation



```
>  initialize_machine(argc,argv);
>  g_sync();
      /* set up */
>     prompt = setup();
>     setup_analyze();

      /* loop over input sets */
>     while( readin(prompt) == 0){
   /* perform warmup trajectories */
>  dtime = -dclock();
   for(traj_done=0; traj_done < warms; traj_done++ ){
>              update();
   }
>  if(this_node==0)printf("WARMUPS COMPLETED\n");

   /* perform measuring trajectories, reunitarizing and measuring  */
   meascount=0;  /* number of measurements  */

>  plp = cmplx(99.9,99.9);
   avm_iters = avs_iters = 0;
```

Step 2: Program Compilation and Execution



Changes from Non-Instrumented Case

- Adjust application's Makefile(s)
  - Replace source code filenames
    e.g. prog.c ? prog.Context.inst.c
  - Compile *InstrumentationInit.c* and link with it
  - Must always instrument main program
  - Link with **`$(SVPABLO)`** and **`$(PAPI)`**
- Execute instrumented executable
- Combine per-task performance files:

```
% SvPabloCombine –o PerfFile c_SDDF*.asc
```

Interface to PAPI Hardware Performance Counters

- User-configured file is read at runtime
  - desired PAPI counters are specified in file
  - if file unavailable, default counter set is used
  - SvPabloLibrary versions: with or without PAPI
  - synthesized metrics: e.g. MFLOPS, % branches mispredicted
- 8 hardware counters are available on Seaborg, to see the available HW events, issue the command:

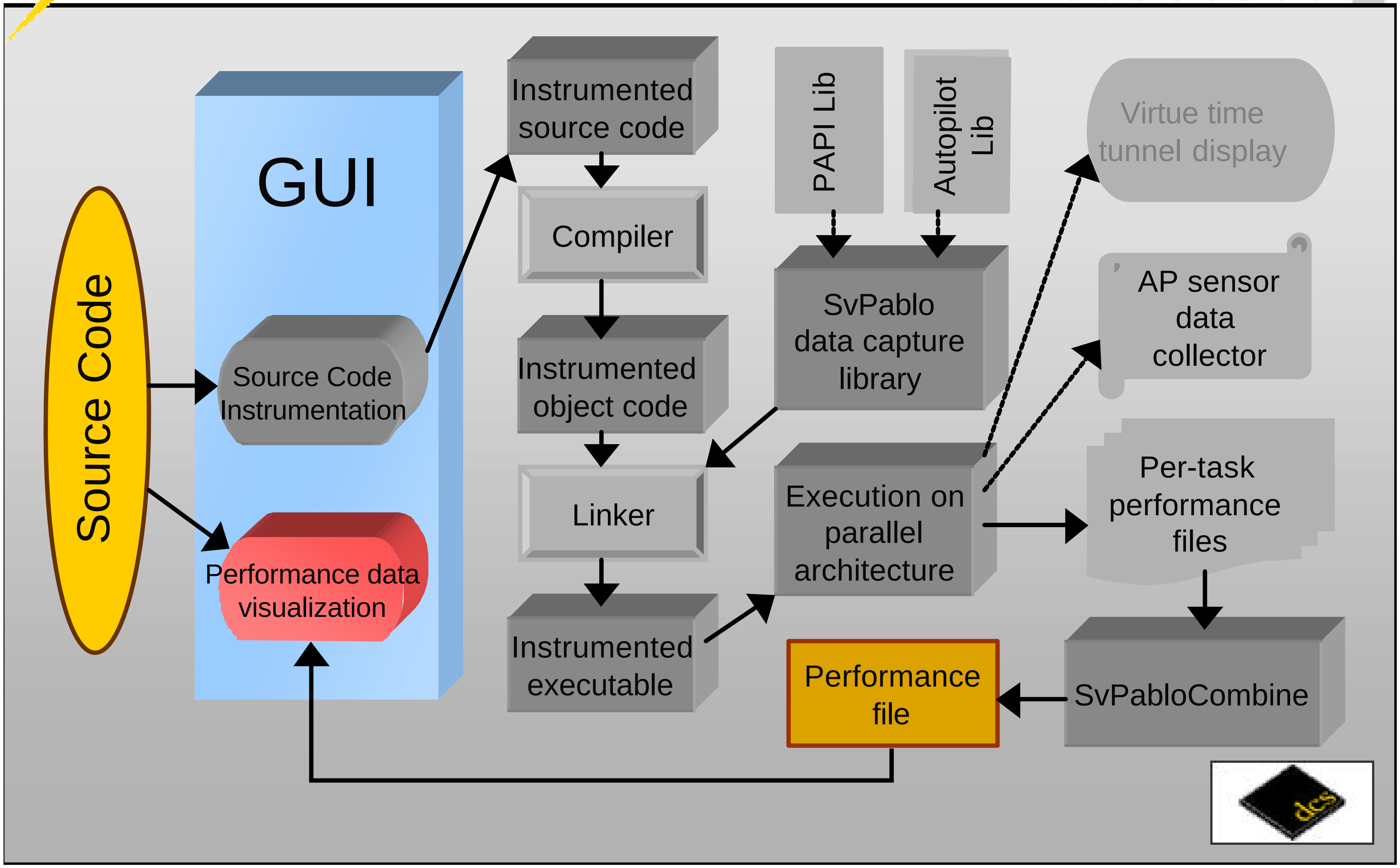
```
% /usr/common/usg/papi/2.1/src/tests/avail
```

See the SvPablo user's guide for details:
ftp://www-pablo.cs.uiuc.edu/pub/Pablo.Release.5/Documentation/SvPabloGuide.ps.gz

NERSC

Step 3: Performance Data Visualization

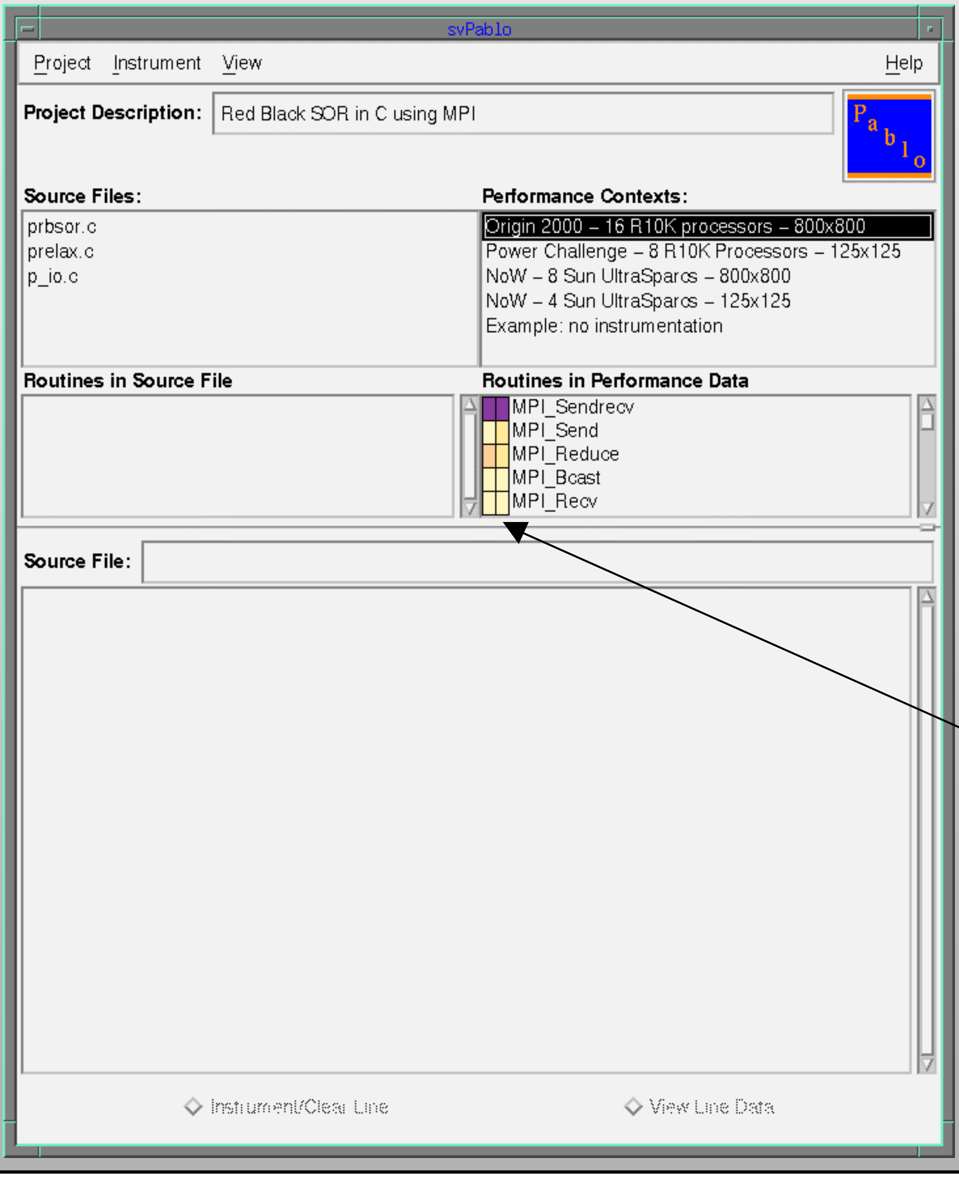
PERC



Performance Data Visualization

- Color encoded GUI
  - configurable by the user
  - no change required to display PAPI data
- Aggregate displays
  - mean and standard deviation values across processors
  - maximum value and its processor number
  - minimum value and its processor number
- Detailed displays
  - individual metric values per processor

Performance Visualization



svPablo

Project Description: Red Black SOR in C using MPI

Source Files:
prbsor.c
prelax.c
p_io.c

Performance Contexts:
Origin 2000 – 16 R10K processors – 800x800
Power Challenge – 8 R10K Processors – 125x125
NoW – 8 Sun UltraSparcs – 800x800
NoW – 4 Sun UltraSparcs – 125x125
Example: no instrumentation

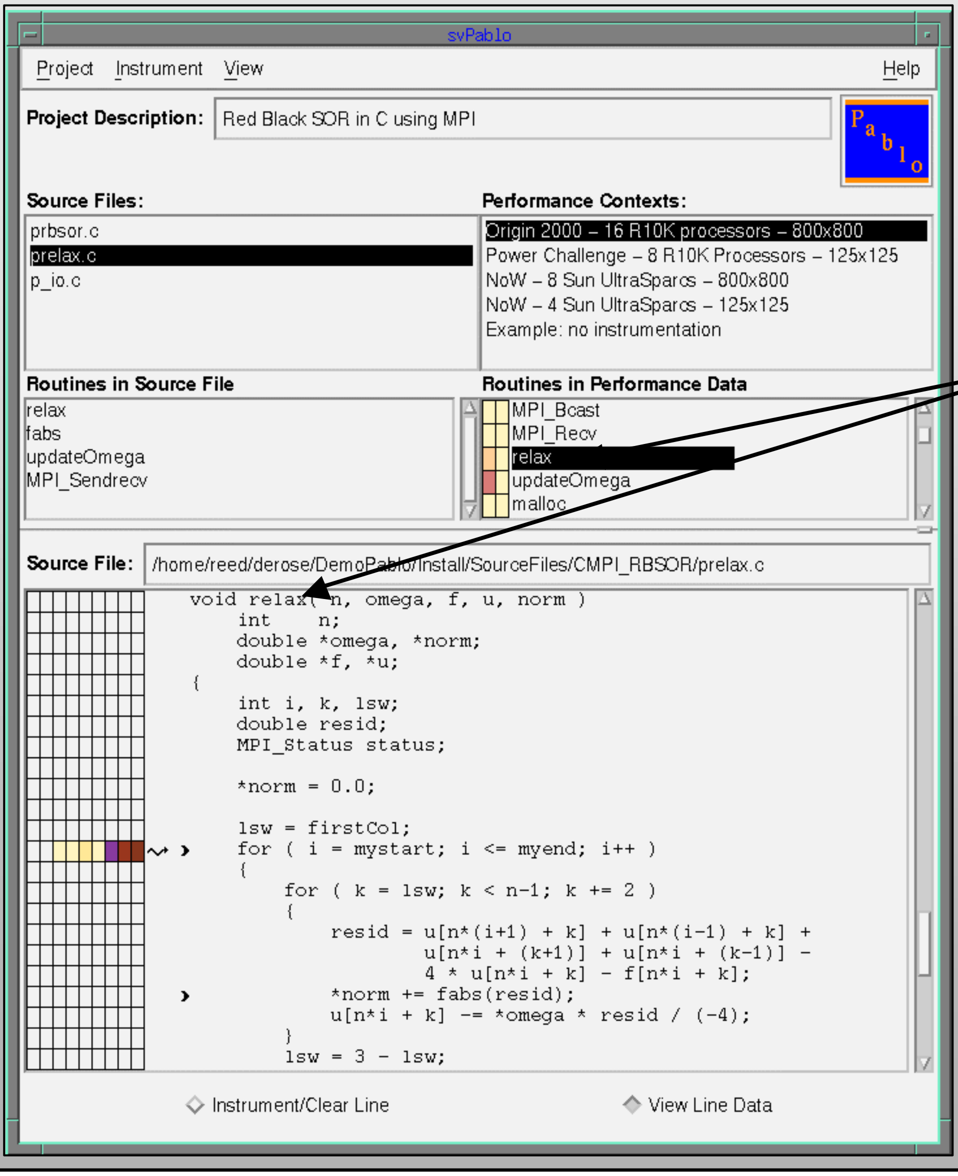
Routines in Source File

Routines in Performance Data
MPI_Sendrecv
MPI_Send
MPI_Reduce
MPI_Bcast
MPI_Recv

Source File:

count, exclusive duration

Function Visualization



Project Description: Red Black SOR in C using MPI

Source Files:
prbsor.c
prelax.c
p_io.c

Performance Contexts:
Origin 2000 – 16 R10K processors – 800x800
Power Challenge – 8 R10K Processors – 125x125
NoW – 8 Sun UltraSparcs – 800x800
NoW – 4 Sun UltraSparcs – 125x125
Example: no instrumentation

Routines in Source File
relax
fabs
updateOmega
MPI_Sendrecv

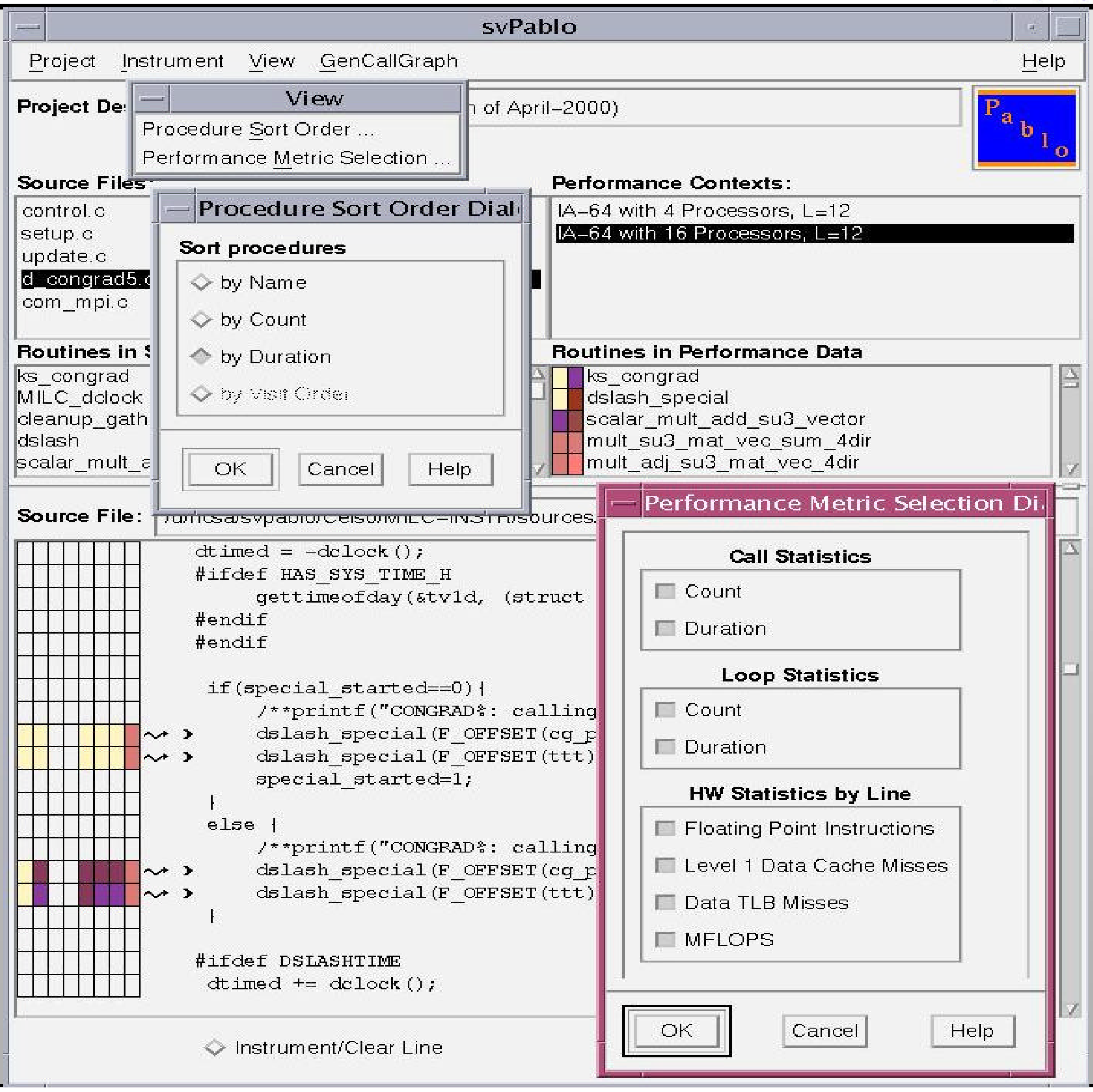
Routines in Performance Data
MPI_Bcast
MPI_Recv
relax
updateOmega
malloc

Source File: /home/reed/derose/DemoPablo/Install/SourceFiles/CMPI_RBSOR/prelax.c

```
void relax( n, omega, f, u, norm )
    int    n;
    double *omega, *norm;
    double *f, *u;
{
    int i, k, lsw;
    double resid;
    MPI_Status status;

    *norm = 0.0;

    lsw = firstCol;
    for ( i = mystart; i <= myend; i++ )
    {
        for ( k = lsw; k < n-1; k += 2 )
        {
            resid = u[n*(i+1) + k] + u[n*(i-1) + k] +
                    u[n*i + (k+1)] + u[n*i + (k-1)] -
                    4 * u[n*i + k] - f[n*i + k];
            *norm += fabs(resid);
            u[n*i + k] -= *omega * resid / (-4);
        }
        lsw = 3 - lsw;
```

selected function

Performance Metric Selection



svPablo

Project Instrument View GenCallGraph Help

Project De

of April–2000)

View

- Procedure Sort Order ...
- Performance Metric Selection ...

Source Files

```
control.c
setup.c
update.c
d_congrad5.c
com_mpi.c
```

Performance Contexts:

- IA–64 with 4 Processors, L=12
- IA–64 with 16 Processors, L=12

Procedure Sort Order Dial

Sort procedures

- by Name
- by Count
- [✓ by Duration]
- by Visit Order

OK Cancel Help

Routines in S

```
ks_congrad
MILC_dclock
cleanup_gath
dslash
scalar_mult_a
```

Routines in Performance Data

```
ks_congrad
dslash_special
scalar_mult_add_su3_vector
mult_su3_mat_vec_sum_4dir
mult_adj_su3_mat_vec_4dir
```

Source File: /u/ncsa/svpablo/Celso/MILC–INSTR/sources

```
dtimed = -dclock();
#ifdef HAS_SYS_TIME_H
     gettimeofday(&tv1d, (struct
#endif
#endif

 if(special_started==0){
     /**printf("CONGRAD%: calling
     dslash_special(F_OFFSET(cg_p
     dslash_special(F_OFFSET(ttt)
     special_started=1;
 }
 else {
     /**printf("CONGRAD%: calling
     dslash_special(F_OFFSET(cg_p
     dslash_special(F_OFFSET(ttt)
 }

#ifdef DSLASHTIME
 dtimed += dclock();
```

Instrument/Clear Line

Performance Metric Selection Di

Call Statistics

- Count
- Duration

Loop Statistics

- Count
- Duration

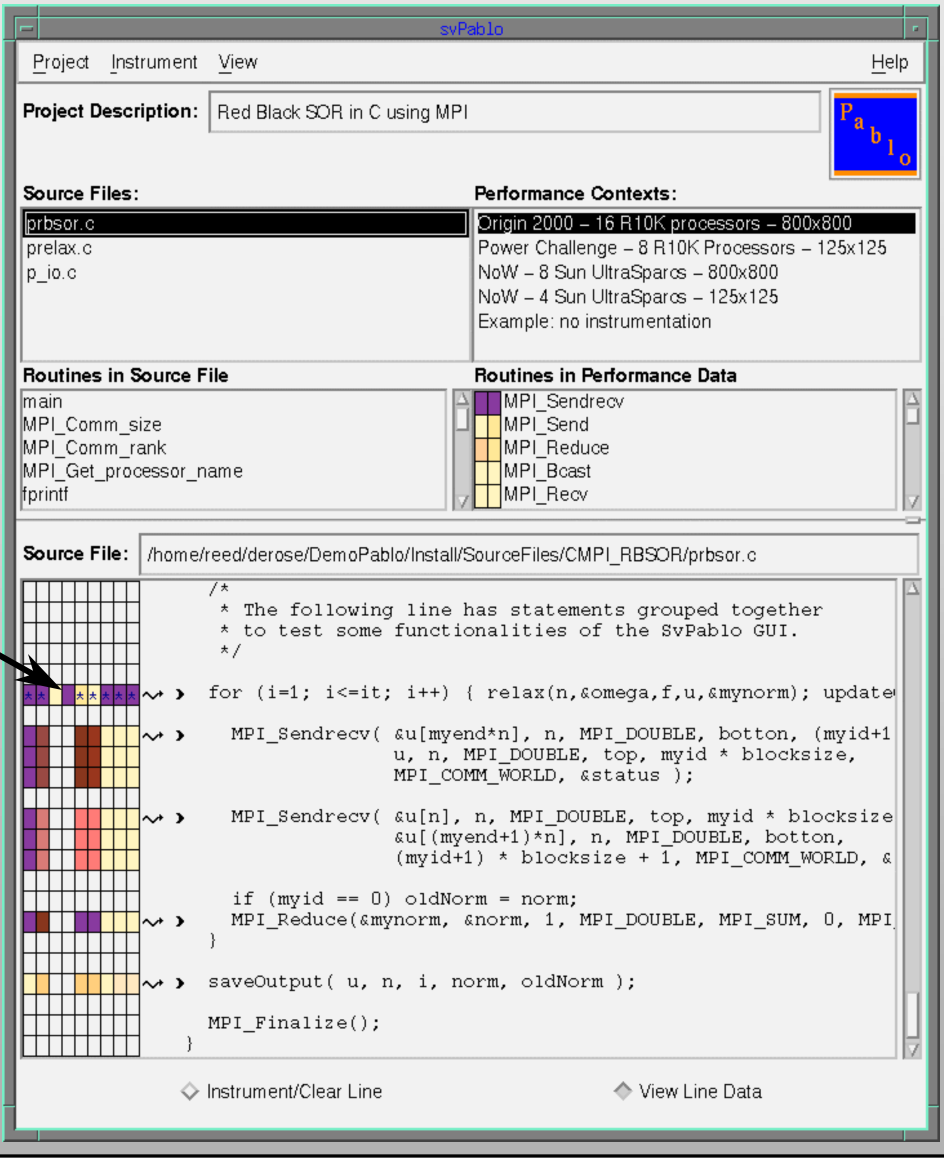
HW Statistics by Line

- Floating Point Instructions
- Level 1 Data Cache Misses
- Data TLB Misses
- MFLOPS

OK Cancel Help

Source Code Visualization

metrics



Project Description: Red Black SOR in C using MPI

Source Files:
prbsor.c
prelax.c
p_io.c

Performance Contexts:
Origin 2000 – 16 R10K processors – 800x800
Power Challenge – 8 R10K Processors – 125x125
NoW – 8 Sun UltraSparcs – 800x800
NoW – 4 Sun UltraSparcs – 125x125
Example: no instrumentation

Routines in Source File
main
MPI_Comm_size
MPI_Comm_rank
MPI_Get_processor_name
fprintf

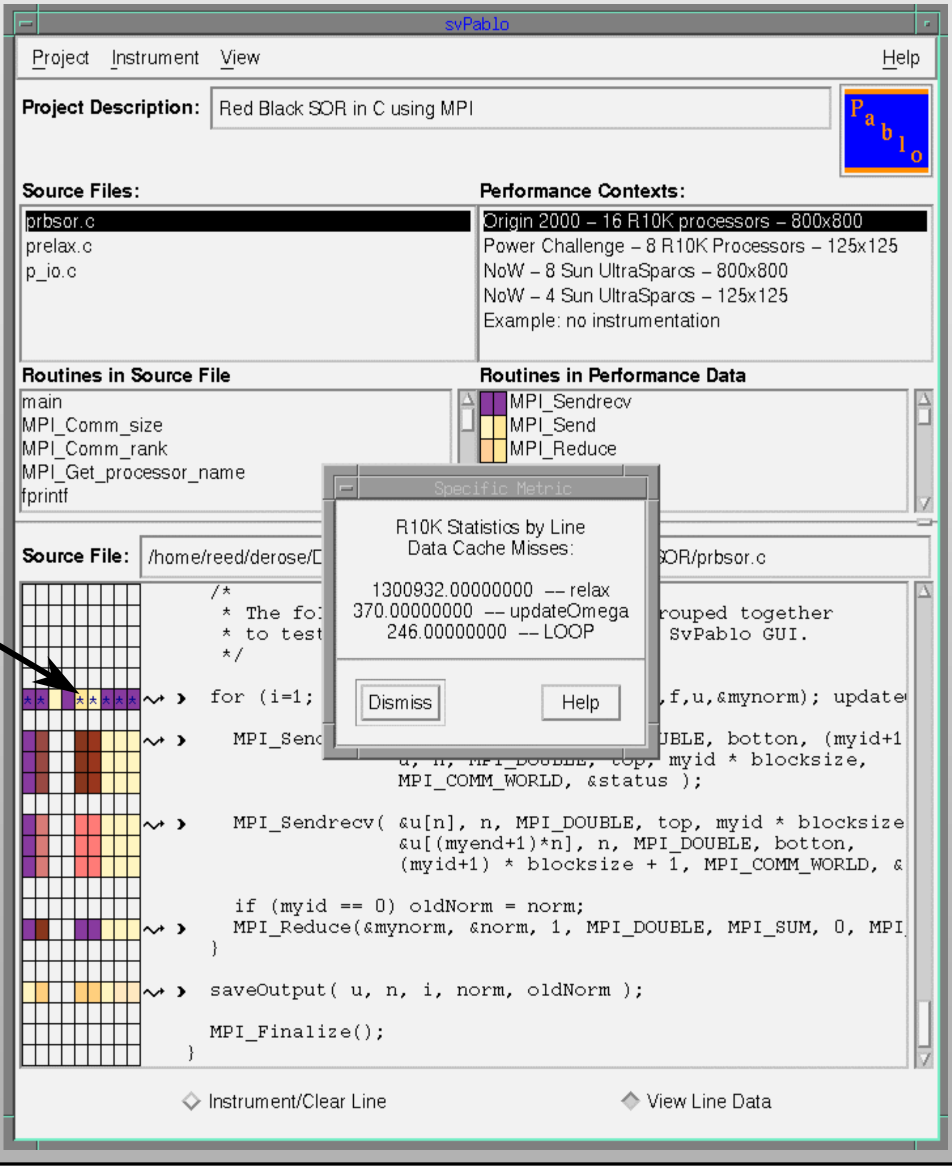
Routines in Performance Data
MPI_Sendrecv
MPI_Send
MPI_Reduce
MPI_Bcast
MPI_Recv

Source File: /home/reed/derose/DemoPablo/Install/SourceFiles/CMPI_RBSOR/prbsor.c

```
      /*
       * The following line has statements grouped together
       * to test some functionalities of the SvPablo GUI.
       */
      for (i=1; i<=it; i++) { relax(n,&omega,f,u,&mynorm); update

        MPI_Sendrecv( &u[myend*n], n, MPI_DOUBLE, botton, (myid+1
                      u, n, MPI_DOUBLE, top, myid * blocksize,
                      MPI_COMM_WORLD, &status );

        MPI_Sendrecv( &u[n], n, MPI_DOUBLE, top, myid * blocksize
                      &u[(myend+1)*n], n, MPI_DOUBLE, botton,
                      (myid+1) * blocksize + 1, MPI_COMM_WORLD, &

        if (myid == 0) oldNorm = norm;
        MPI_Reduce(&mynorm, &norm, 1, MPI_DOUBLE, MPI_SUM, 0, MPI
      }

      saveOutput( u, n, i, norm, oldNorm );

      MPI_Finalize();
    }
```

Multiple Events Metric Box

mark for multiple events



Project Description: Red Black SOR in C using MPI

Source Files:
- prbsor.c
- prelax.c
- p_io.c

Performance Contexts:
- Origin 2000 – 16 R10K processors – 800x800
- Power Challenge – 8 R10K Processors – 125x125
- NoW – 8 Sun UltraSparcs – 800x800
- NoW – 4 Sun UltraSparcs – 125x125
- Example: no instrumentation

Routines in Source File
- main
- MPI_Comm_size
- MPI_Comm_rank
- MPI_Get_processor_name
- fprintf

Routines in Performance Data
- MPI_Sendrecv
- MPI_Send
- MPI_Reduce

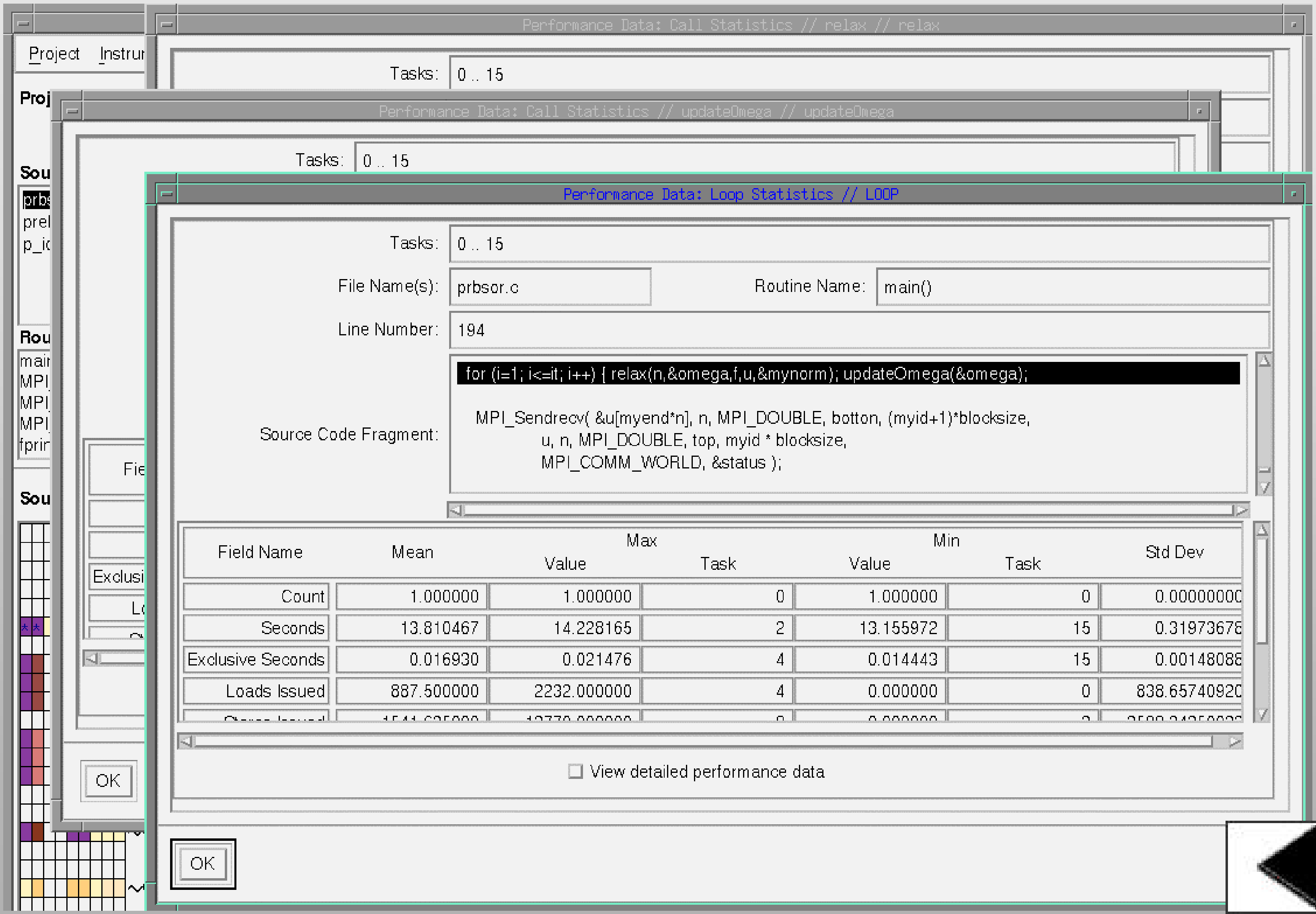
R10K Statistics by Line
Data Cache Misses:

1300932.00000000 -- relax
370.00000000 -- updateOmega
246.00000000 -- LOOP

Source File: /home/reed/derose/D … OR/prbsor.c

```
  /*
   * The fo…                                    …rouped together
   * to test…                                    SvPablo GUI.
   */

  for (i=1; …                                  …,f,u,&mynorm); update

    MPI_Send…                                  …UBLE, botton, (myid+1
                          u, n, MPI_DOUBLE, top, myid * blocksize,
                          MPI_COMM_WORLD, &status );

    MPI_Sendrecv( &u[n], n, MPI_DOUBLE, top, myid * blocksize
                  &u[(myend+1)*n], n, MPI_DOUBLE, botton,
                  (myid+1) * blocksize + 1, MPI_COMM_WORLD, &

    if (myid == 0) oldNorm = norm;
    MPI_Reduce(&mynorm, &norm, 1, MPI_DOUBLE, MPI_SUM, 0, MPI
  }

  saveOutput( u, n, i, norm, oldNorm );

  MPI_Finalize();
}
```

Performance Statistics



Performance Data: Call Statistics // relax // relax

Tasks: 0 .. 15

Performance Data: Call Statistics // updateOmega // updateOmega

Tasks: 0 .. 15

Performance Data: Loop Statistics // LOOP

Tasks: 0 .. 15

File Name(s): prbsor.c

Routine Name: main()

Line Number: 194

Source Code Fragment:

```
for (i=1; i<=it; i++) { relax(n,&omega,f,u,&mynorm); updateOmega(&omega);

  MPI_Sendrecv( &u[myend*n], n, MPI_DOUBLE, botton, (myid+1)*blocksize,
          u, n, MPI_DOUBLE, top, myid * blocksize,
          MPI_COMM_WORLD, &status );
```

Field Name	Mean	Max Value	Max Task	Min Value	Min Task	Std Dev
Count	1.000000	1.000000	0	1.000000	0	0.00000000
Seconds	13.810467	14.228165	2	13.155972	15	0.31973678
Exclusive Seconds	0.016930	0.021476	4	0.014443	15	0.00148088
Loads Issued	887.500000	2232.000000	4	0.000000	0	838.65740920
Stores Issued	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

☐ View detailed performance data

The Roadmap for the PERC Tool Suite

Currently:

- Tools are being ported to more HPC platforms.
- Additional functionality is being added to the tools.
- Application groups are using the tools and providing feedback.

Future:

- Port tools to all major HPC platforms.
- Improve functionality with the help of feedback from application groups.
- Enhance the functionality of the tools and work on interoperation.
- Integrate performance models with the tools.

✍ The HPC community will be presented with a robust, versatile, and portable suite of performance tools that is suited to a modern HPC environment.

The Roadmap for the PERC Tool Suite

PAPI

- Multi-way multiplexing
- Faster substrates

Dyninst

- Implement full functionality on all platforms
- Build infrastructure for use with parallel applications

Sigma++

- Predict Performance using trace data
- Compile-time instrumentation for collection of data-dependence information

Performance Bounds

- Build on infrastructure for C/C++ and Fortran-77 source code
- Automate the application of performance bounding techniques

SvPablo

- Improve infrastructure for supporting Fortran-90 codes
- Develop an infrastructure for supporting C++ codes
- Add support for OpenMP

Interoperate with the other tools and instrumentation systems:

Dyninst: Use the Dyninst API to allow dynamic instrumentation and analysis at runtime

Performance Bounding: Use to display performance bounds alongside measured performance

Sigma: Enable Sigma instrumentation and visualization of Sigma results